Investigation of KVM Network Performance

Simon Horman

14th March 2018

Contents

1 Introduction

The purpose of this investigation is to evaluate the performance of networking from KVM guests to the host of the guests and onwards to remote hosts.

The document describes the forwarding mechanisms evaluated, the environment used for testing, the tests run, and presents results of these tests.

Guest–Host Forwarding

Four different mechanisms of forwarding packets from a guest to the host of the guest have been evaluated.

- **User Networking**

  provides a simple mechanism to connect guests and allow communication by them to both host they are running on and remote hosts. As the name suggests the implementation runs in user-space, on the host running the guest. It provides NAT to allow the guest to access hosts and thus by default hosts cannot access the guest. A limitation of this mechanism is that it only forwards UDP and TCP.

- **Emulated e1000**

  providees a tap device to the host running the guest and a e1000 device to the guest. The guest may communicate using a standard e1000 driver and thus does not require modification. The host may forwards packets between the guest and remote host using layer 3 routing or layer 2, typically using some form of bridge.

- **Virtio**

  is a paravirtualised approach which offers significant performance advantages over emulated devices, however, the guest needs to have a virtio networking driver. Like emulated devices the host running the guest has a tap device and may forward packets using layer 2 or layer 3 mechanisms.

- **VhostNet**

  is an enhancement to Virtio whcih offers further significant performance advantages by using the Linux kernel rather than QEMU to transmit packets between the guest and the host.

Host–Remote Forwarding

Two mechanisms for forwarding packets between guests and remote hosts via the host of the guest were evaluated:

- **User Networking**

  provides forwading using layer 3 in conjundtion with built-in NAT of guests.

- **Bridge**, in this case the Linux Bridge, allows layer 2 forwarding between network devices. In this case two devices are used, the tap device connected to the guest and a physical Ethernet device.

2 Environment

Host of Guests

The host of the guests used has a single 4 core/8 thread; i7-3770 CPU @ 3.40GHz with 16Gb RAM at a r8169 NIC. It is running Debian GNU/Linux 9.4 and its 4.9.0-6-amd64 kernel.

In order to eliminate any effect of recently added Spectre fixes on performance they were turned off by passing `spectre_v2=off` on the kernel command line. *This is not a recommended configuration for production environments*.

The netperf package was installed for testing described in this document.

Remote Host

The remote host used has a single 4 core/4 thread; i3-4010U CPU @ 1.70GHz with 4Gb RAM and an e1000e NIC. It was not CPU bound during the course of any tests presented in this document.

It is running Debian GNU/Linux 9.4 and its 4.9.0-6-amd64 kernel.

The netperf package was installed for testing described in this document.

Guest

The guest is a KVM instance involved from the command line using the KVM command supplying a kernel image, kernel command line parameters, initrd to act as a minimal user-space and networking options. The -nographic option presents the VMs console to the terminal where it is invoked. The VMs are given 128M of memory.

As per the discussion of CPU pinning later in this document, KVM us invoked using taskset to pin the processes to designated CPU threads.

For example:

```
# taskset 11 kvm \
    -m 128 \
    -kernel ./bzImage \
    -append "console=ttyS0 spectre_v2=off" \
    -initrd ./rootfs.cpio.gz \
    -netdev type=tap,id=net0 \
    -device e1000,netdev=net0 \
    -nographic
```

Kernel

The guest kernel is Linux v4.16-rc2 compiled using the kernel configuration provided separately.

As for the host of the guests in order to eliminate any effect of recently added Spectre fixes on performance they were turned off by passing `spectre_v2=off` on the kernel command line. *This is not a recommended configuration for production environments*.

Buildroot

The initrd for the guests is buildroot 2018.02-rc2 compiled using the configuration provided separately.

CPU Pinning

Some consideration was given to pinning tasks to CPUs to allow better performance. The result of that experimentation is that the following setting were used when running the tests described in this document.

- User Networking
  - KVM (and qemu-system-x86_64) on thread x of core A
  - For testing forwarding to host of the guest: netserver on same thread
  - For testing forwarding to remote host: eth0 IRQ (there is only one) on thread y of core A
- Virtio without VhostNet:
  - KVM (and qemu-system-x86_64) on 2 threads of 1 core
  - For testing forwarding to host of the guest: netserver on the same 2 threads
  - For testing forwarding to remote host: eth0 IRQ (there is only one) on one thread of the same core
- Virtio with VhostNet
  - KVM (and qemu-system-x86_64) on 1 or 2 threads of core A
  - vhost on thread y of core B
  - For testing forwarding to host of the guest: netserver on thread x of core B
  - For testing forwarding to remote host: eth0 IRQ (there is only one) on thread y of core B
- Emulated e1000:
  - KVM (and qemu-system-x86_64) on 1 or 2 threads of core A (2)
  - For testing forwarding to host of the guest: netserver on thread x of core B (1)
  - For testing forwarding to remote host: eth0 IRQ (there is only one) on thread x of core B

3 Tests and Results

Guest–Host–Guest

The purpose of this test is to evaluate forwarding performance between the guest and host without overhead from forwarding by the host of the guest to and from a remote host.

Test

Netperf was used to evaluate forwarding performance. A STREAM_UDP test was used to evaluate raw packet forwarding without interference from congestion control, window scaling and so on found in TCP.

The host of the guest was prepared for the test as follows. The br0 portion of the setup is not invoked in this case as there is no bridge in use.

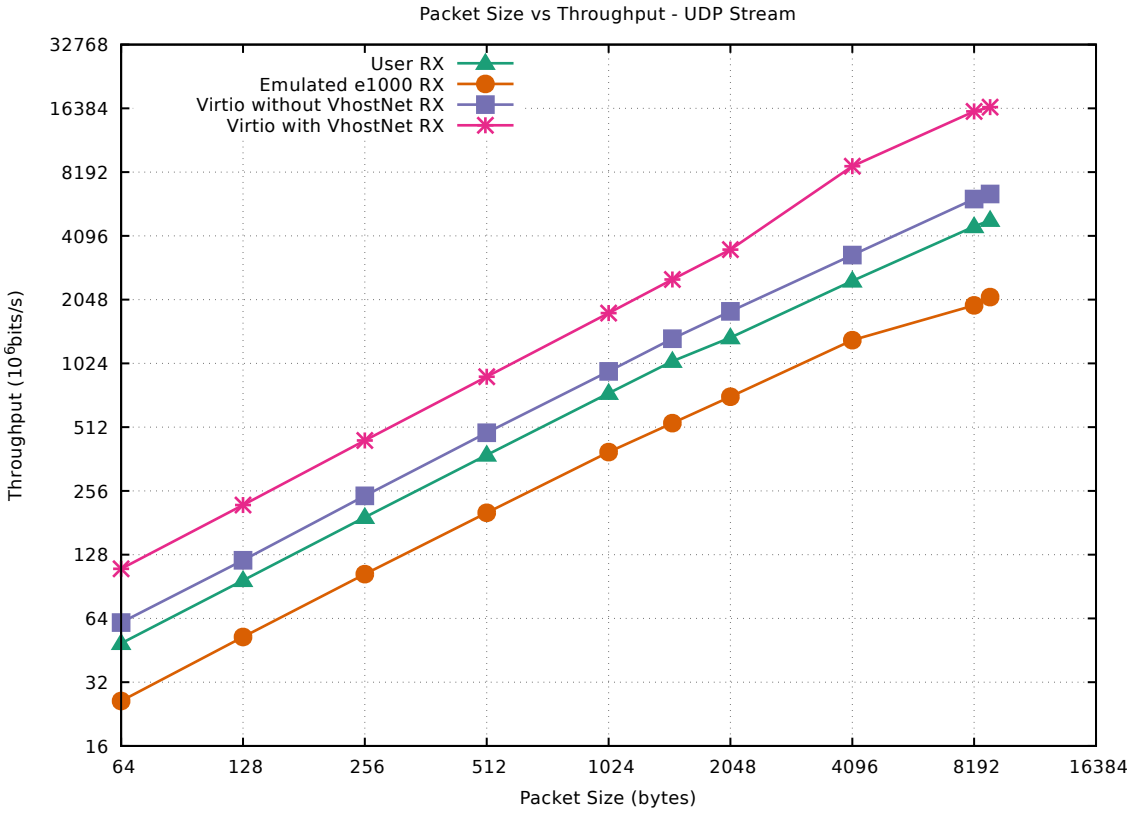
```
#!/bin/sh

do_taskset_p ()
{
    MASK="$1"; shift

    for i in "$@"; do
        taskset -p $MASK $i
    done
}

for i in $(grep eth0 /proc/interrupts | cut -f 1 -d :); do
    echo 4 > /proc/irq/$i/smp_affinity_list
done

if ! ip link sh  | grep tap0: > /dev/null; then
    # User Networking
    do_taskset_p 1 $(pidof netserver)
    exit
fi

if ! ip link sh br0 > /dev/null; then
    ip addr add 10.0.99.2/24 dev tap0
    ip link set mtu 9000 dev tap0
    ip link set up dev tap0
fi

VHOST_PIDS="$(pidof vhost-$(pidof qemu-system-x86_64))"
if [ -n "$VHOST_PIDS" ]; then
    # vhost-net
    do_taskset_p 10 $(pidof vhost-$(pidof qemu-system-x86_64))
    do_taskset_p 1 $(pidof netserver)
else
    # virtio or emulated
    do_taskset_p 11 $(pidof netserver)
fi
```

The test was initiated on the guest as it throughput was observed to be bounded in that direction, typically by CPU.

On the guest the test was invoked as follows:

```
#!/bin/sh

ip addr add 10.0.99.192/24 dev eth0
ip link set mtu 9000 dev eth0
ip link set up dev eth0

for i in 64 128 256 512 1024 1472 2048 4096 8192 8972; do
    netperf -H 10.0.99.2 -t UDP_STREAM -i 20,2 -I 99,20 -c 100 -C 100 -- \
        -m $i -s 32768 -S 32768
    sleep 10
done
```

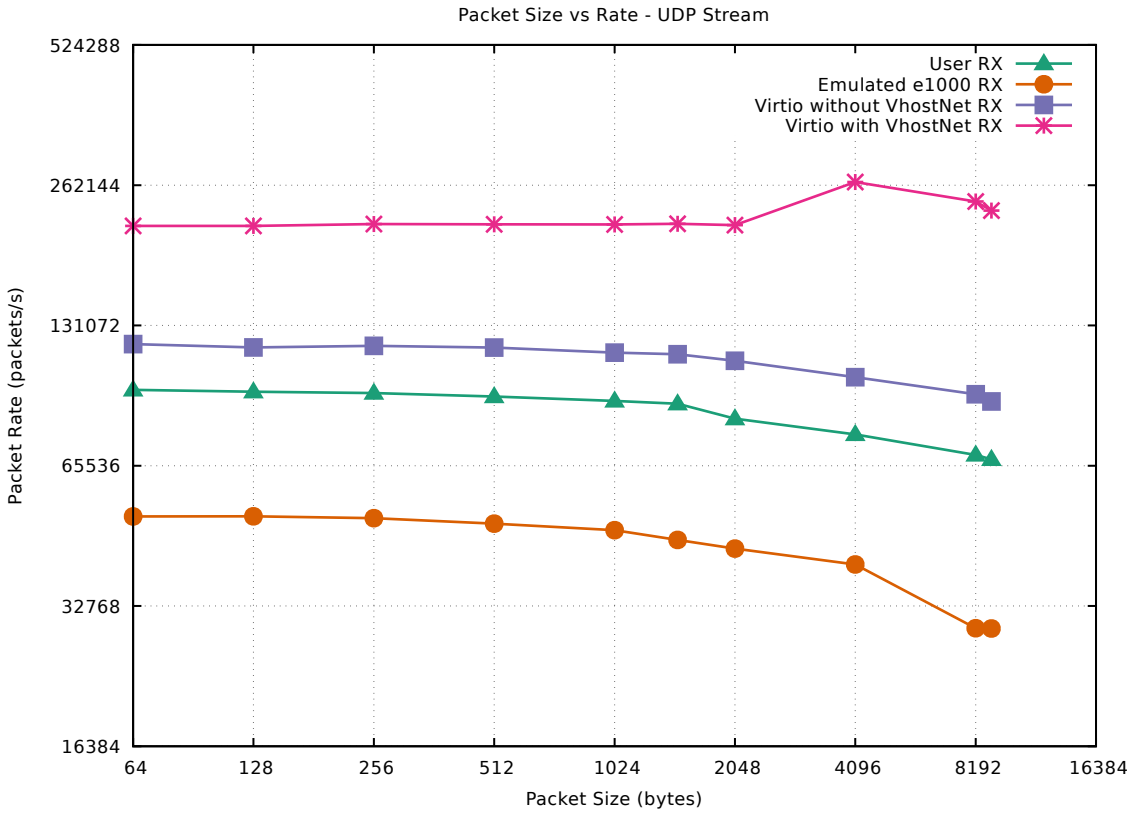
Results



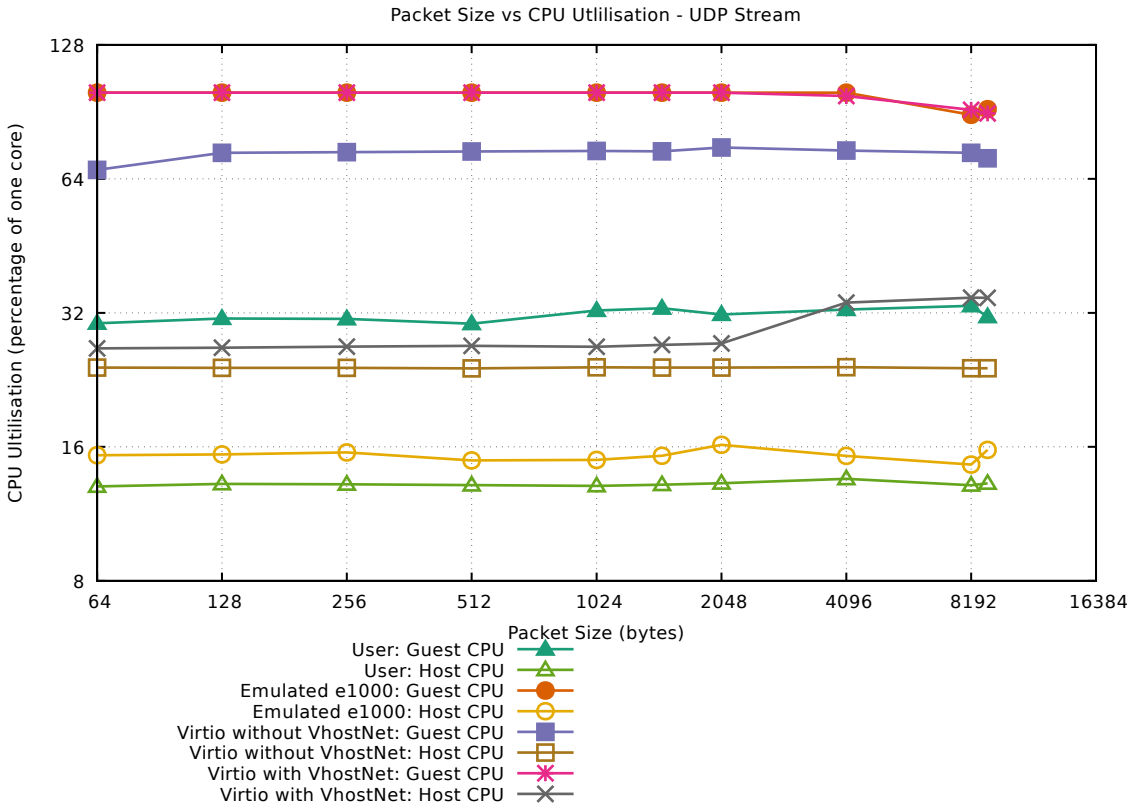
Please note that as with all graphs presented the scale of both axis is logarithmic.

The results show that for all packet sizes Virtio with VhostNet gives the best throughput followed by Virtio without VhostNet, emulated e1000 and user networking. And that throughput increases with packet size indicating the influence of CPU availability on the results.

For both throughput and rate only RX is shown, RX results were very close and would only clutter the graph.



The packet rate declines slightly as packet size increases indicating that the limitation on throughput is not entirely due to per-packet processing overhead. There exception to this trend is that for Virtio with VhostNet the packet rate for 4096 byte packets is greater than for 2048. This appears to be an anomaly which bears further investigation.



For each forwarding mechanism host and guest CPU utilisation is more or less consistent across packet sizes.

Both Virtio with VhostNet and emulated e1000 consumes 100% of a guest CPU core and significantly less on the host. The combined host and guest CPU usage of Virtio without VhostNet, which is pinned to two threads of a single core to maximise throughput is 100%. It appears that all three of these forwarding mechanisms are CPU bound in this test.

The combined host and guest CPU utilisation of User Networking is well below 100% indicating that this forwarding mechanism is not CPU bound in this test.

Guest–Remote–Guest Routed on Host

The purpose of this test is to evaluate forwarding performance between the guest and a remote host. The host of the guests routes packets two and from the user networking instance.

Virtio and Emulated forwarding between a guest and its host were not part of this test. Rather they were exercised using bridging as described later in this document.

Test

As per the Guest–Host–Guest test a netperf STREAM_UDP test was used to measure throughput, packet rate and CPU utilisation.

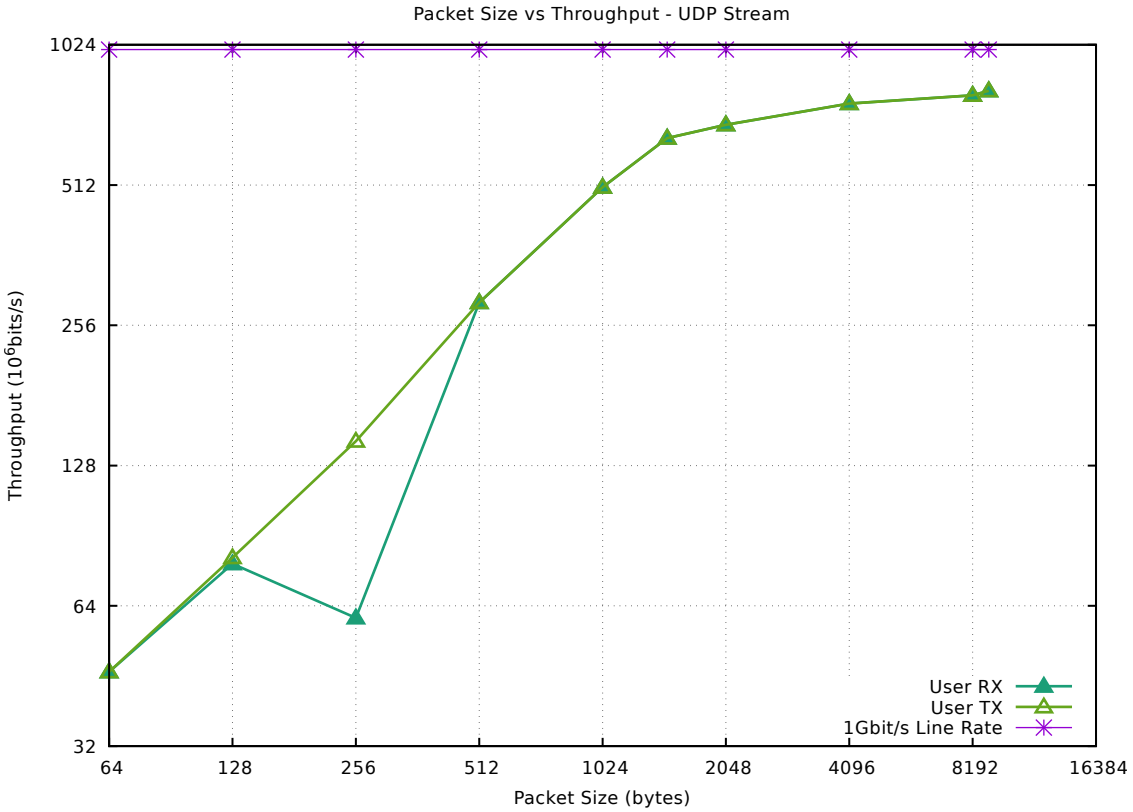
The host of the guest was prepared for the test as per the preparation for the Guest–Host–Guest test.

The test was initiated on the guest as it throughput was observed to be bounded in that direction, typically by CPU.

On the guest the test was invoked as follows:

```
#!/bin/sh

ip addr add 10.0.99.192/24 dev eth0
ip link set mtu 9000 dev eth0
ip link set up dev eth0

for i in 64 128 256 512 1024 1472 2048 4096 8192 8972; do
    netperf -H 10.4.3.2 -t UDP_STREAM -i 20,2 -I 99,20 -c 100 -C 100 -- \
        -m $i -s 32768 -S 32768 -R 1
    sleep 10
done
```

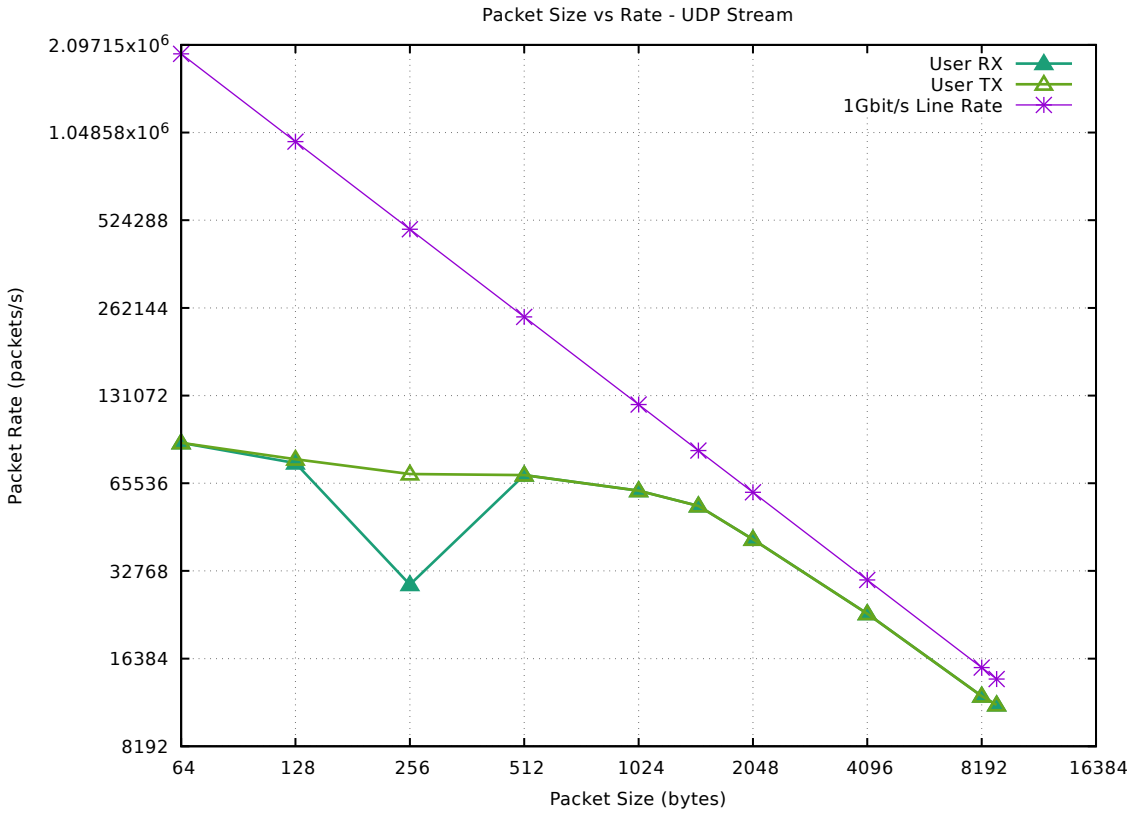
Results

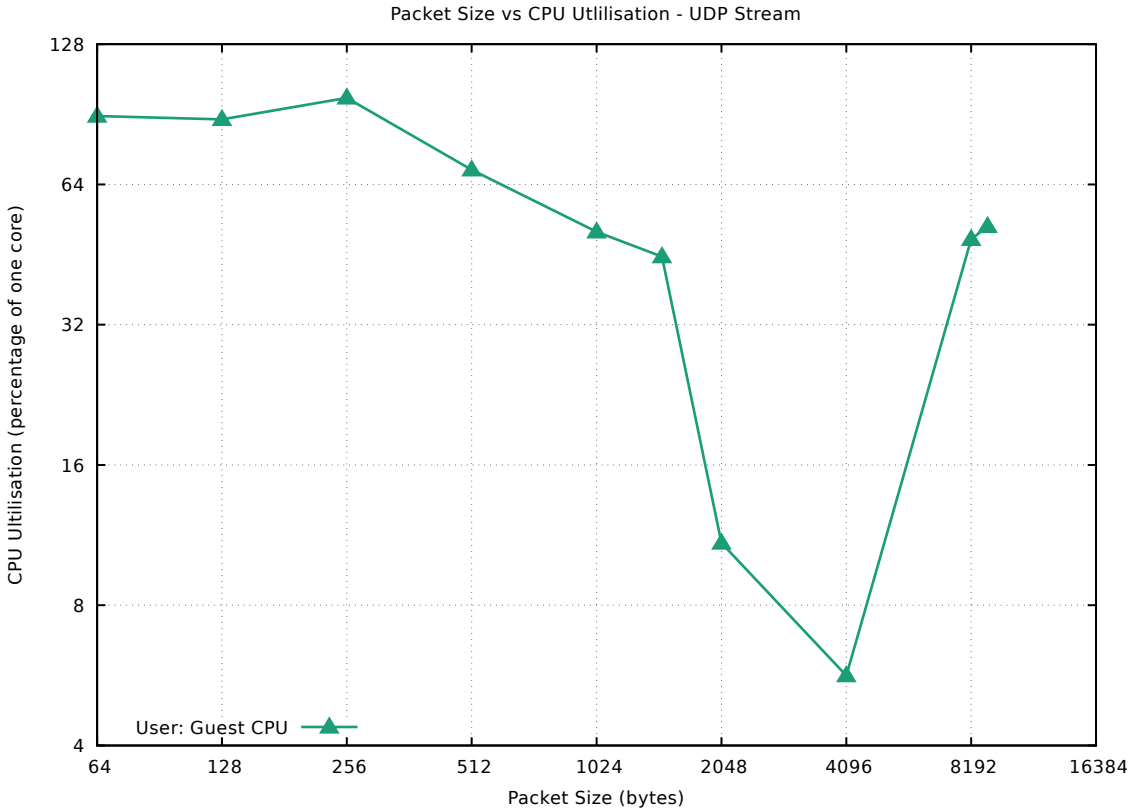


Throughput and packet rate approaches line rate as packet sizes approach 9000 bytes. There is an anomaly for 256 byte packets whereby RX speed is significantly slower than that of 128 byte packets.

In general the guest CPU utilisation declines with packet sizes, however, there are several anomolies in the data including a small rise in CPU utulisation for 256byte packets and adramatic drop for 2048 and 2096 byte packets. The former corresponds to an anomily in the RX rate at 256 bytes. While the drop at 2048 and 4096 bytes and sbsequent dramatic climb at 8192 bytes may relate to ring-buffer layout or the need for higher-order memory allocations to handle larger packets in the host running the guest. This bears further investigation.

For both throughput and rate only RX as shown, RX results were very close and would only clutter the graph.


Packet Size vs Rate - UDP Stream
Packet Rate (packets/s)
2.09715x10^6
1.04858x10^6
524288
262144
131072
65536
32768
16384
8192
64
128
256
512
1024
2048
4096
8192
16384
Packet Size (bytes)
User RX
User TX
1Gbit/s Line Rate


Packet Size vs CPU Utlilisation - UDP Stream
CPU Ultilisation (percentage of one core)
128
64
32
16
8
4
64
128
256
512
1024
2048
4096
8192
16384
Packet Size (bytes)
User: Guest CPU

Bridged Guest–Remote–Guest Bridged on Host

The purpose of this test is to evaluate forwarding performance between the guest and a remote host. The tap interface of the host-side of a host-guest link is added to a Linux bridge along with the physical ethernet interface of the host of the guest. This allows layer 2 forwarding between guests and remote hosts.

Virtio and Emulated e1000 are used in conjunction with the bridge. User networking is tested using routing to forward packets to remote hosts as described in the previous section of this document.

Test

As per tests described earlier in this document a netperf STREAM_UDP test was used to measure throughput, packet rate and CPU utilisation.

The host of the guest was prepared for the test as per the preparation for the Guest–Host–Guest test.

The test was initiated on the guest as it throughput was observed to be bounded in that direction, typically by CPU.

On the guest the test was invoked as per the Guest–Host–Guest test.

On the host running the guest a bridge, br0, was created: the hosts physical interface, eth0, was added to the bridge, and the local IP address and associate routes were moved from eth0 to br0 to allow the host to communicate with both the guest and remote hosts. This configuration was achieved as follows.

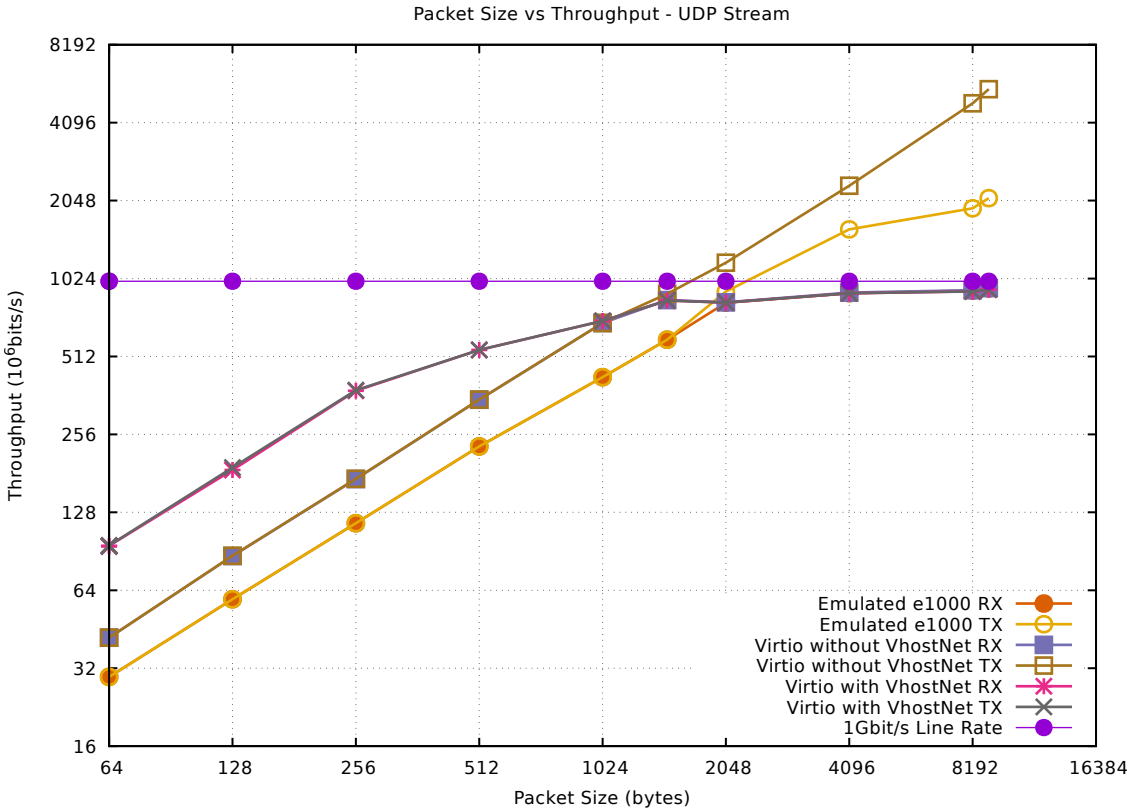
```
#!/bin/sh

BR_DEV="br0"
UPPER_DEV="eth0"
ADDR=10.4.3.162/24
GW=10.4.3.2

brctl addbr $BR_DEV
ip link set up $BR_DEV
ip link set up dev $UPPER_DEV

brctl addif $BR_DEV $UPPER_DEV
ip addr del dev $UPPER_DEV $ADDR
ip addr add dev $BR_DEV $ADDR
ip route add 0.0.0.0/0 via $GW
```

KVM will typically automatically add the host-side interface of the host-guest link to br0. If not it may be done as follows, in this case the interface is tap0.

```
brctl addif br0 tap0
```

Further configuration of the host was as per that run for the Guest–Host–Guest test, this time the br0 portion of that setup is invoked.

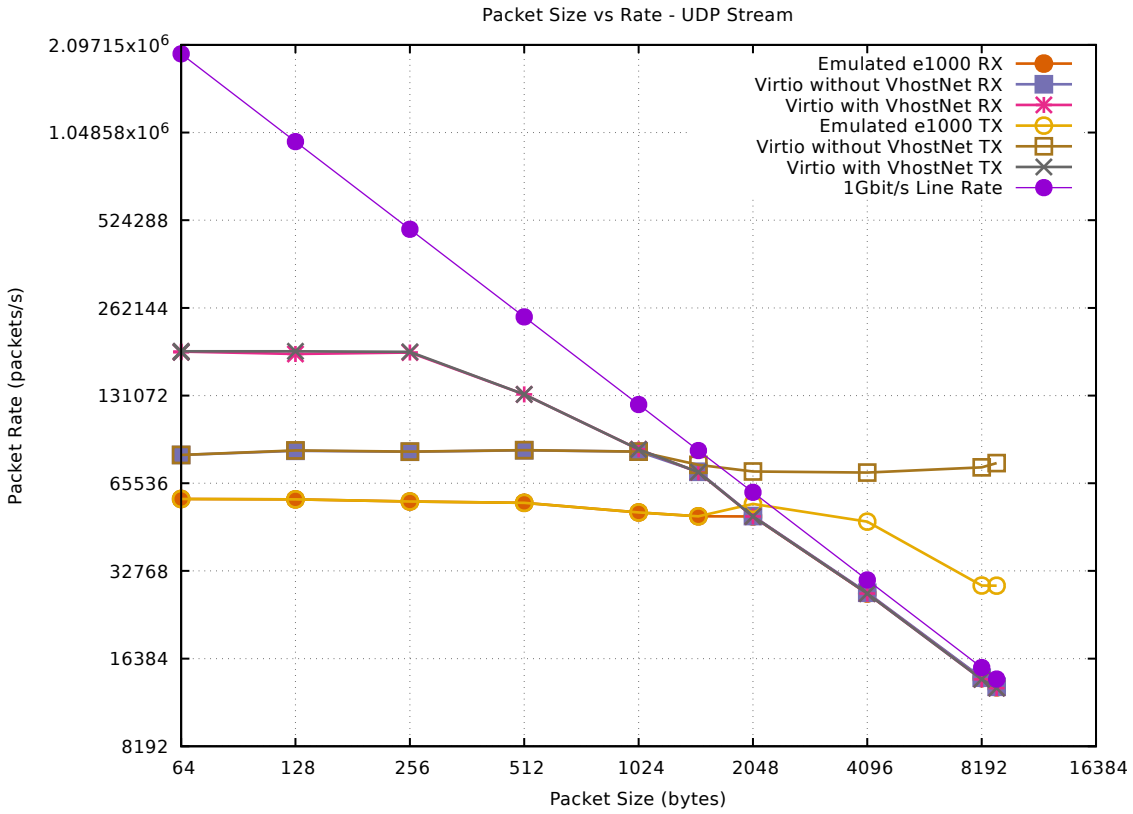
Results

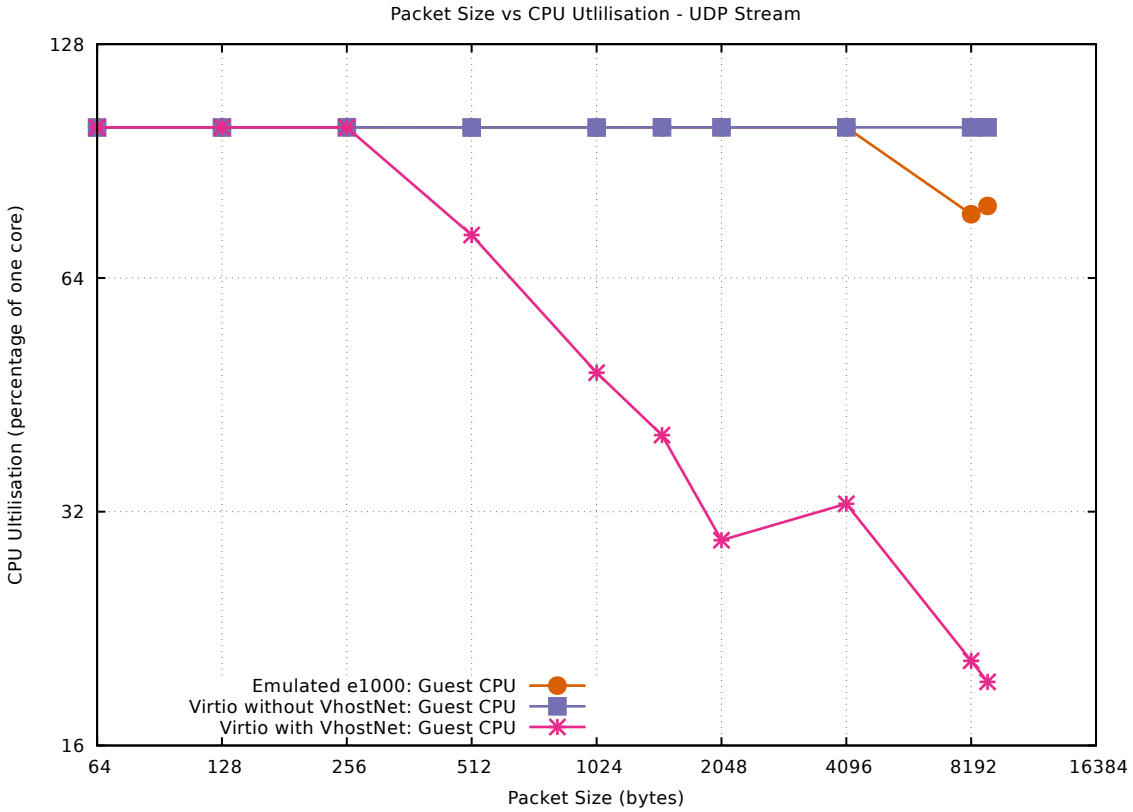


For packet sizes less than 1024 bytes Virtio with VhostNet gives the best throughput followed by Virtio without VhostNet and Emulated e1000.

For Virtio with and without VHostNet at a packet size of 1024 bytes RX throughput becomes bounded by the 1Gbit/s line rate of of the physical Ethernet interface of the host running the guest. This occurs at a packet size of 1472 bytes for Emulated e1000.

At this line-rate limit it appears that there is a back-pressure mechanism for Virtio with VhostNet which also limits transmit to 1Gbit/s. Neither Virtio without VhostNet nor Emulated e1000 exhibit back-pressure and are able to transmit at greater than the line rate of the host, the excess packets are dropped in the host.


Packet Size vs Rate - UDP Stream
Packet Rate (packets/s)
Packet Size (bytes)
2.09715x10^6
1.04858x10^6
524288
262144
131072
65536
32768
16384
8192
64
128
256
512
1024
2048
4096
8192
16384
Emulated e1000 RX
Virtio without VhostNet RX
Virtio with VhostNet RX
Emulated e1000 TX
Virtio without VhostNet TX
Virtio with VhostNet TX
1Gbit/s Line Rate


Packet Size vs CPU Utlilisation - UDP Stream
CPU Ultilisation (percentage of one core)
Packet Size (bytes)
128
64
32
16
64
128
256
512
1024
2048
4096
8192
16384
Emulated e1000: Guest CPU
Virtio without VhostNet: Guest CPU
Virtio with VhostNet: Guest CPU

CPU utilisation for on the guest for Virtio without VHostNet is consistent at 100% for all packet sizes. This indicates that it is CPU bound and that in the absence of any back-pressure TX will consume as much CPU as is available.

Virtio with VHostNet appears to be CPU bound up until 256 byte packets and beyond that packet size CPU consumption drops dramatically. This appears to be consistent with back-pressure resuding the rate at wich packets are sent in accordance with limitations of the Ethernet device on the host running the guest.

Emulated e1000 is CPU bound up until 4096 byte packets after which there is a small drop in guest CPU utisation. The drop at this point, which also corresponds to a drop in packet rate and flattening of the increase in throughput may relate to extra work being done on the host – for instance relating to the ring-buffer management or the need for higher-order memory allocations. This bears further investigation

4 Conclusion

Virtio with VHostNet consistently offers the best performance in terms of throughput, packet rate and CPU utilisatin. It also appears to have a back-pressure mechanism that avoids sending packets to the host it is running on at a rate faster than it can forward them, thus avoiding wasteful CPU utilisation.

Following Virtio with VHostNet, Virtio without VHostNet gives the best performance followed by Emulated e1000 and finally User Networking.